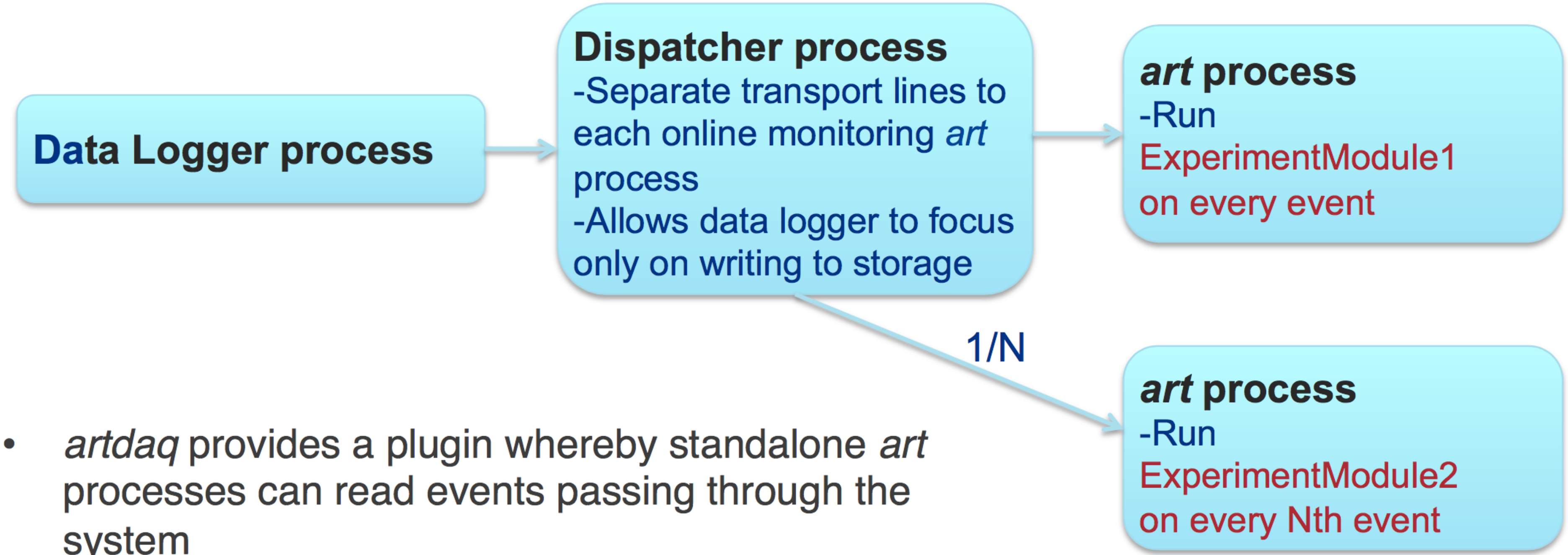
Online Monitoring

John Freeman
10 January 2018

Online Monitoring in *artdaq*

- The basic model is the ability to launch an art process which can read in live events sent to it by an artdaq dispatcher process
- Lots of f lexibility is built-in:
  - As long as the dispatcher is initialized, it's possible to launch online monitoring processes and kill them at any point without affecting the primary DAQ dataf low
  - In principle, can have any number of dispatchers on any number of nodes, with any number of art processes attached to each dispatcher
  - Can apply art modules both in the online monitoring art processes (primarily f lter and analysis modules) as well as on the Dispatcher end (primarily f lter modules)
- Robust – online monitoring art crashes will not affect primary DAQ dataf low

Online Physics Monitoring



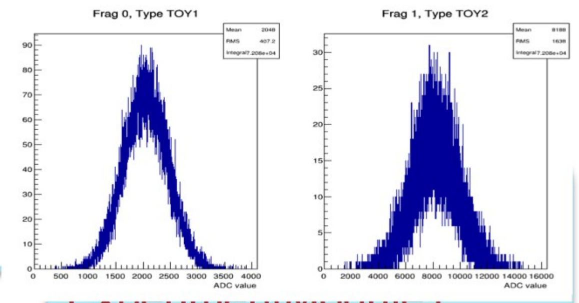
Data Logger process
Dispatcher process
-Separate transport lines to each online monitoring *art* process
-Allows data logger to focus only on writing to storage
art process
-Run ExperimentModule1 on every event
1/N
art process
-Run ExperimentModule2 on every Nth event

- *artdaq* provides a plugin whereby standalone *art* processes can read events passing through the system
- Can configure fraction of events sent to a process, or even apply experiment-specific cuts!

Online Physics Monitoring

Data Logger process

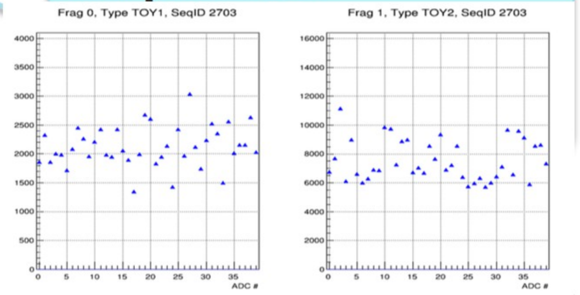
Dispatcher process
-Separate transport lines to each online monitoring *art* process
-Allows data logger to focus only on writing to storage



on every event

1/N

***art* process**



- *artdaq* provides a plugin whereby standalone *art* processes can read events passing through the system
- Can configure fraction of events sent to a process, or even apply experiment-specific cuts!

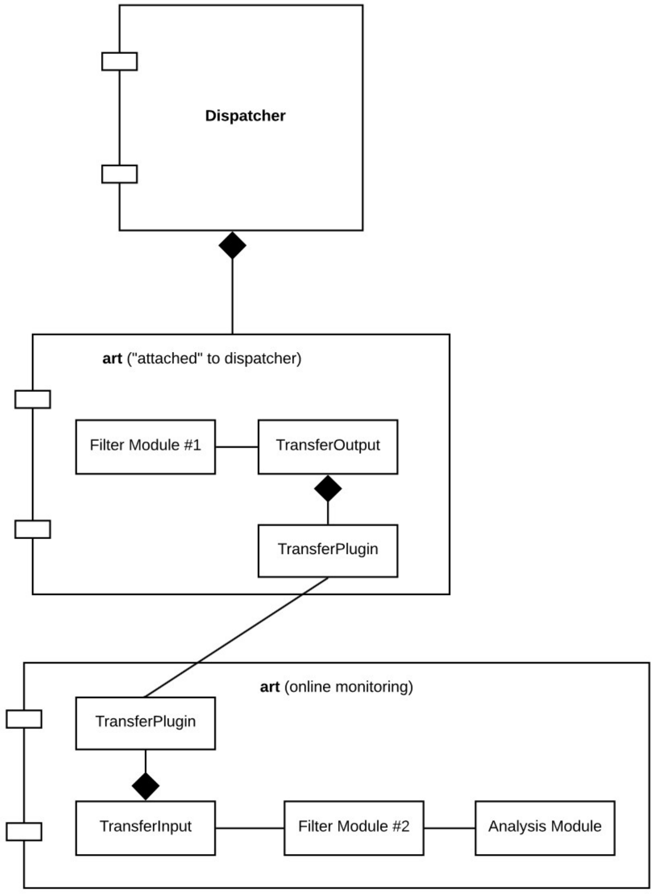
Fermilab

When an Online Monitoring Process Is Launched

- artdaq supplies an input source for art, TransferInput, which is capable of reading in live events from the DAQ system (cf. RootInput, which reads in events from *.root files)
- TransferInput sends the FHiCL definition of a transfer plugin and art modules up to the dispatcher. The art modules are optional, and would typically be filter modules.
- The dispatcher launches a “sister” art process which includes the art modules in its workflow, and uses the transfer plugin definition in its TransferOutput output to send events to the online monitoring art process

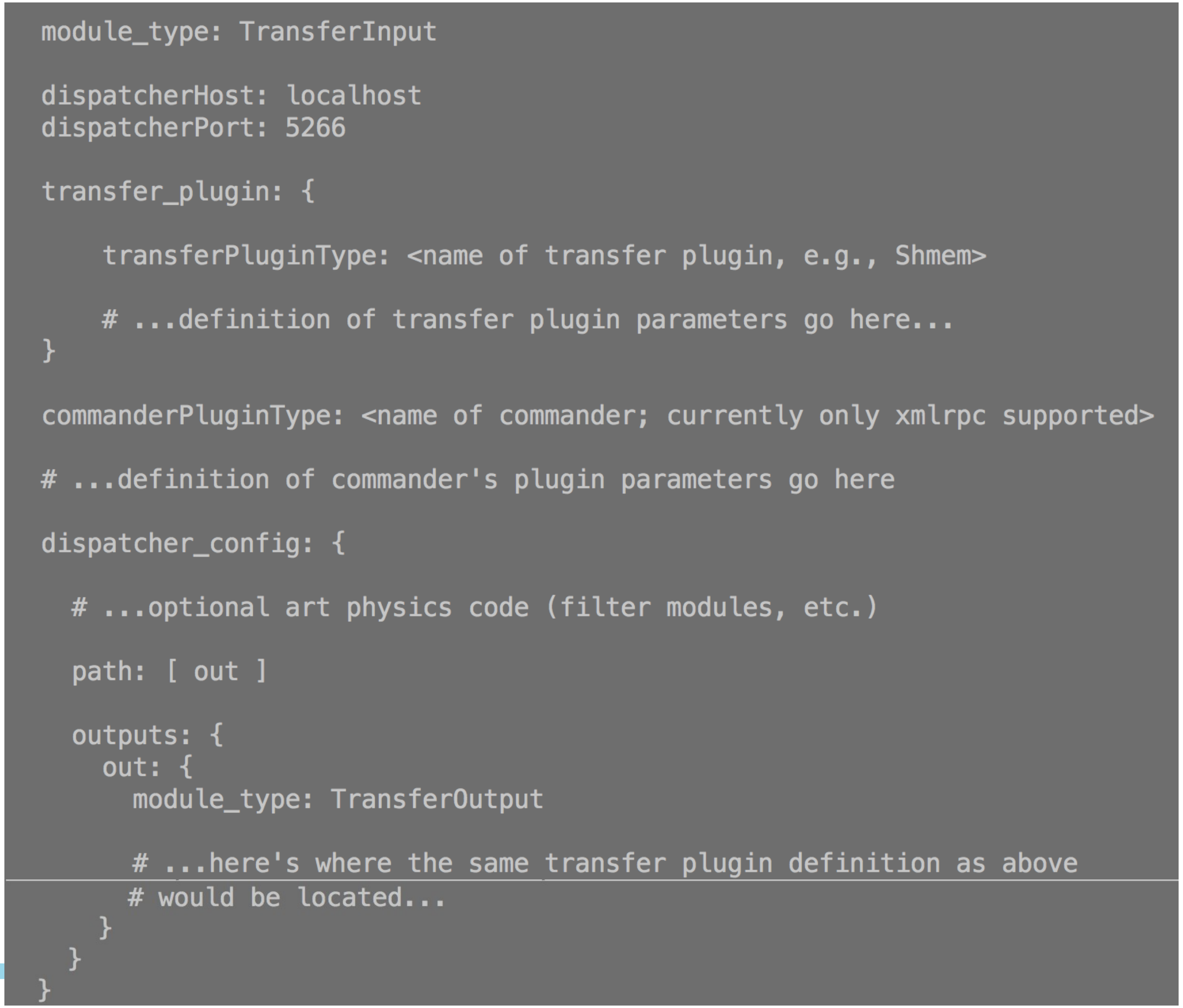
What This Looks Like

- This is a UML diagram of an art online monitor connected to a Dispatcher process
- Notched rectangle == process
- Normal rectangle == class instance
- Line without diamond == “works with”
- Line with diamond == “owns”



The FHiCL

- This is a skeleton, in that I've omitted some of what would actually be here so we don't get lost in details
- The dispatcher_config table is sent up to the Dispatcher via the commander plugin so that it can use the contents to start the sister art process



```
module_type: TransferInput

dispatcherHost: localhost
dispatcherPort: 5266

transfer_plugin: {

    transferPluginType: <name of transfer plugin, e.g., Shmem>

    # ...definition of transfer plugin parameters go here...
}

commanderPluginType: <name of commander; currently only xmlrpc supported>

# ...definition of commander's plugin parameters go here

dispatcher_config: {

  # ...optional art physics code (filter modules, etc.)

  path: [ out ]

  outputs: {
    out: {
      module_type: TransferOutput

      # ...here's where the same transfer plugin definition as above
      # would be located...
    }
  }
}
```

Stopping Online Monitoring

- To stop online monitoring, one needs only to send the art process the SIGINT signal – i.e., Ctrl-c in its terminal, or “kill -SIGINT <art process ID>”, as is done in /nfs/sw/om/fcl/StopOM_lite.sh on the teststand

- As mentioned before, this will not affect datataking – events will still flow through the system if it's in the “running” state

- When SIGINT is received, TransferInput sends a message to the Dispatcher telling it that the art online monitor wants to unregister; the Dispatcher will kill off the sister art process

- If the online monitoring dies a messy death, then the sister art process will continue to exist and attempt to send events into the “ether”. Again, the primary DAQ dataflow will be unaffected.

- Related – if we're running and the sister art process dies a messy death, the Dispatcher will regenerate it!

A Boundary Condition

- In a terminal, when setting up the environment to perform online monitoring, not only are the packages used in the art workflow needed but artdaq (and its dependencies) as well, as artdaq supplies TransferInput
- Therefore, any packages which both artdaq and the art modules depend on have to have the same versions and qualifiers
- On protoDUNE, we use the dunetpc package, which, like artdaq, is based on artdaq-core. Unlike artdaq, its artdaq-core version on the develop branch bumps up relatively frequently, and outside our control, introducing version mismatches.

Ensuring Consistent Package Versions

- Whenever dunetpc bumps their artdaq-core version, we bump ours
  - This would require frequent new releases of artdaq, with all the overhead that implies. Plus protoDUNE's not the only experiment which would be bumping off lne package versions
- Move TransferInput into artdaq-core
  - At least currently, this would effectively couple every experiment's off lne software to our online software, introducing things like a dependency on XML-RPC which don't make sense in an off lne context
- Have a feature branch of dunetpc dedicated to online monitoring which we (myself, Jingbo, Tonino, etc.) can control: feature/online_monitoring_artdaq
  - Issue here is that useful new online code isn't immediately available to off lne, and vice versa. Plus if useful new off lne code is made using a version of artdaq-core which isn't compatible with the one on the online feature branch, we've got some handshaking issues

Also on the Subject of Packaging

- It's not guaranteed that art modules which run in an experiment's online monitor art process will be able to run in the monitor's upstream sister art process
- E.g., on protoDUNE, while online monitoring uses dunetpc, dune-artdaq doesn't depend on this package
- This may not be a big issue to the extent that in general you're more likely to need a fuller set of packages for online monitoring purposes than for cutting-at-the-Dispatcher-level purposes
- Therefore, any packages which both artdaq and the art modules depend on have to have the same versions and qualifiers

In Summary

- A lot of flexibility is available in terms of the art workflows you can apply to events for online monitoring purposes
- Flexibility is also available for the physical transport mechanism of events (transfer plugins)
- And there's even (potential) flexibility in how an art monitoring process sends the Dispatcher its FHiCL code
- Sins are (mostly) forgiven: very slow art modules and even art modules which crash won't slow down the “primary path” of events, getting written to disk by the DataLogger(s)
- Although it's not a perfect world: package versions must be carefully managed